Java: Generics

IF2210 – Semester II 2020/2021

by: RSP; rev: SAR

Pengantar

- Pada proyek pengembangan perangkat lunak sering muncul bug. Dengan perencanaan, *programming*, dan *testing* yang baik akan mereduksi munculnya bug.
- Ada bug yang lebih mudah dideteksi yaitu *compile-time bug* (dibandingkan *run-time bug*).
- Dengan menggunakan konsep Generik akan menambah stabilitas kode dengan membuat bug terdeteksi saat kompilasi.

Contoh: simple box

```
public class SimpleBox {
    private Object object;
    public void put(Object object) { this.object = object; }
    public Object get() { return object; }
}

public class SimpleBoxDemo1 {

    public static void main(String[] args) {

        // ONLY place Integer objects into this box!
        SimpleBox intBox = new SimpleBox();

        intBox.put(10);

        int someInt = (int) intBox.get();
        System.out.println(someInt);
    }
}
```

```java
public class SimpleBoxDemo2 {

    public static void main(String[] args) {

        // ONLY place Integer objects into this box!
        SimpleBox intBox = new SimpleBox();

        intBox.put("10"); // Someone may mistakenly put a String...

        int someInt = (int) intBox.get();
        System.out.println(someInt);
    }
}
```

Menyebabkan *run-time exception*:

```
Exception in thread "main"
  java.lang.ClassCastException:
    class java.lang.String cannot be cast to class java.lang.Integer
    at SimpleBoxDemo2.main(SimpleBoxDemo2.java:10)
```

Generic pada Java

- Kelas generik pada java diimplementasikan sebagai parameter tipe.
- Hasil kompilasi kode kelas generik tetap hanya satu kelas, dengan parameter tipe yang diganti dengan tipe riil pada saat *runtime*.
- pada C++, kompilasi *class template* menghasilkan kelas yang berbeda untuk setiap tipe generik.
- Keuntungan generik:
  - Meningkatkan *expressive power*.
  - Meningkatkan *type safety*.
  - Mengeksplisitkan parameter tipe dan mengimplisitkan *type casting*.

Sintaks Generik

- Mendefinisikan kelas & interface:
  - `class NamaKelas<TipeGenerik> { ... }`
  - `interface NamaInterface<TipeGenerik> { ... }`
  - `class NamaKelas<TipeGenerik1, TipeGenerik2> { ... }`
  - `interface NamaInterface<TipeGenerik1, TipeGenerik2> { ... }`
- Nama untuk parameter tipe generik sebaiknya menggunakan sebuah karakter huruf besar, misalnya `E` atau `T`.
- Kita dapat mendefinisikan lebih dari satu parameter tipe.

Konvensi Penamaan Tipe

- E - Element (used extensively by the Java Collections Framework)
- K - Key
- N - Number
- T - Type
- V - Value
- S, U, V etc. - 2nd, 3rd, 4th types

Contoh: generic box

```
public class Box<T> {
    private T t;
    public void put(T t) { this.t = t; }
    public T get() { return t; }
}

public class GenericBoxDemo1 {
    public static void main(String[] args) {

        Box<Integer> intBox = new Box<>(); // "diamond notation" for ctor
         intBox.put(10);

          int someInt = intBox.get(); // no casting needed
          System.out.println(someInt);
     }
}
```

```
public class GenericBoxDemo2 {

    public static void main(String[] args) {

        Box<Integer> intBox = new Box<>();

        intBox.put("10");

        int someInt = intBox.get();
        System.out.println(someInt);
    }
}
```

Menyebabkan *compile error*:

```
Error:(7, 20) java: incompatible types: java.lang.String cannot be
converted to java.lang.Integer
```

Generic methods & constructors

- *Type parameter* dapat juga digunakan pada *method* dan konstruktor menjadi *generic methods* dan *generic constructors*.

```
public class Box<T> {
    // ...
    public <U> void inspect(U u) {
        System.out.println("T: " + t.getClass().getName());
        System.out.println("U: " + u.getClass().getName());
    }
    // ...
}

// Somewhere else...
Box<Integer> intBox = new Box<>();
intBox.put(10);
intBox.inspect("10");
```

Type inference

- *Compiler* dapat mengetahui tipe parameter pada pemanggilan method generik dari tipe argumennya.

```
public class Box<T> {
    // ...
    public static <U> void fillBoxes(U u, List<Box<U>> boxes) {
        for (Box<U> box: boxes) box.put(u);
    }
    // ...
}

// Somewhere else...
Crayon red = new Crayon("red");
List<Box<Crayon>> crayonBoxes = new ArrayList<>();

Box.<Crayon>fillBoxes(red, crayonBoxes);
Box.fillBoxes(red, crayonBoxes); // compiler infers that U is Crayon
```

Bounded type parameters

- Ada kebutuhan untuk membatasi tipe-tipe apa saja yang diberbolehkan untuk masuk sebagai parameter.
- Sebagai contoh mengharapkan hanya tipe angka (`Number` dan turunannya) yang boleh.

```
public class Box<T> {
    // ...
    public <U extends Number> void boundedInspect(U u) {
        System.out.println("T: " + t.getClass().getName());
        System.out.println("U: " + u.getClass().getName());
    }
    // ...
}
```

```java
public class BoundedTypeParamDemo {

    public static void main(String[] args) {

        Box<Integer> intBox = new Box<>();
        intBox.put(10);
        intBox.boundedInspect("10"); // error, String does not
                                     // extend Number
    }
}
```

Menyebabkan *compile error*:

```
Error:(7, 15) java: method boundedInspect in class Box<T> cannot be
applied to given types;
  required: U
  found: java.lang.String
  reason: inference variable U has incompatible bounds
    lower bounds: java.lang.Number
    lower bounds: java.lang.String
```

Bound dengan interface

- Tetap menggunakan *keyword* `extends` (bukan `implements`).
- Jika lebih dari satu, gabungkan dengan operator &.
- Contoh:

```
public <T extends SomeInterface> void a(T t) {...}
public <U extends Number & SomeInterface> void b(U u) {...}
```

Subtyping (1)

- Ingat kembali relasi "*is a*". Jika `D extends` atau `implements` `B`, maka semua *reference* ke `B` dapat di-*assign* dengan objek dari kelas `D`.
- Contoh: `Integer extends Number`, maka dapat dilakukan:

```
Number aNumber;
Integer anInt = new Integer(10);
aNumber = anInt; // OK
```

Subtyping (2)

- Berlaku juga untuk pemanggilan method:

```
public void someMethod(Number n) {/*...*/}

// somewhere else...
someMethod(new Integer(10));  // OK
someMethod(new Double(10.0)); // OK
```

- dan pada tipe generik:

```
Box<Number> box = new Box<>();
box.put(new Integer(10));  // OK
box.put(new Double(10.0)); // OK
```

Subtyping (3)

- Bolehkah:

```
public void doSomethingToBoxOfNumber(Box<Number> b) {
    /*...*/
}

// somewhere else:
Box<Integer> intBox = new Box<>();
Box<Double> dblBox = new Box<>();
doSomethingToBoxOfNumber(intBox);
doSomethingToBoxOfNumber(dblBox);
```

  ?

- Akan dibahas pada materi *wildcard*.

Type erasure (1)

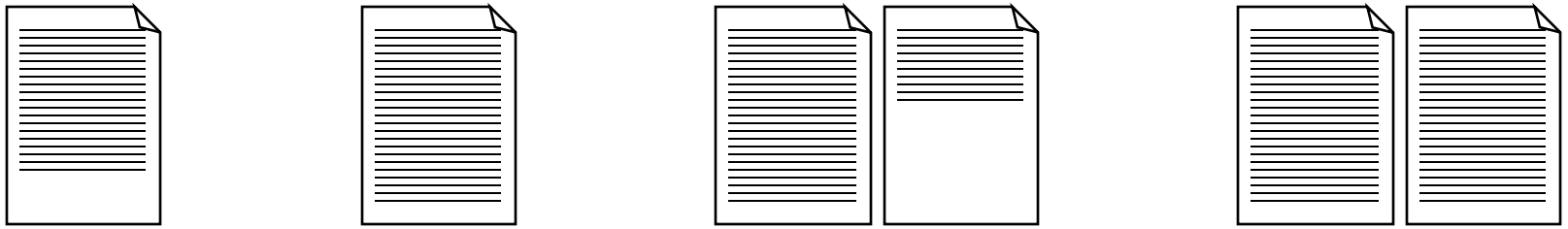
- Bila tipe generik digunakan maka kompilator akan membuang semua informasi yang berhubungan dengan tipe parameter dalam kelas atau *method*.
  - T digantikan dengan *bound*-nya atau `Object`, jika *unbounded*.
  - Ditambahkan *typecasting*.
  - Dibangkitkan *method* antara sebagai jembatan.
- Satu kelas generik pada *source code* (`.java`) hanya dikompilasi menjadi satu kelas dalam *bytecode*, berbeda dari *template* di C++, C# yang membangkitkan kelas-kelas untuk setiap parameter *template* yang digunakan.

Tugas Baca #8A

- Tugas: baca artikel-artikel berikut terkait *type erasure*:
  - code.iamkate.com/articles/java-generics-type-erasure/
  - stackoverflow.com/questions/31693/what-are-the-differences-between-generics-in-c-sharp-and-java-and-templates-i

Tugas Baca #8B

- Gast, H. (2015). *How to use objects: code and concepts*. Addison-Wesley Professional.
- Chapter 3 (semua)
- Buat summary min. 1 halaman, max. 3 halaman
  - excluding header e.g. nama, NIM
  - excluding baris kosong e.g. Enter 2×
- Pengumpulan: Google Classroom



Not OK OK OK OK